LAWRENCE
LIVERMORE
NATIONAL
LABORATORY

UCRL-CONF-235206

User Interface Framework for the National Ignition Facility (NIF)

J. M. Fisher, G. A. Bowers, R. W. Carey, S. A. Daveler, K. B. Herndon Ford, J. C. Ho, L. J. Lagin, C. J. Lambert, J. Mauvais, E. A. Stout, S. L. West

October 2, 2007

International Conference on Accelerator and Large Experimental Physics Control Systems
Knoxville, TN, United States
October 14, 2007 through October 20, 2007

Disclaimer

USER INTERFACE FRAMEWORK FOR THE NATIONAL IGNITION FACILITY (NIF)*

J. Fisher, G. Bowers, R. Carey, S. Daveler, K. Herndon Ford,
J. Ho, L. Lagin, C. Lambert, J. Mauvais, E. Stout, S. West
Lawrence Livermore National Laboratory, Livermore, CA, U.S.A.

Abstract

A user interface (UI) framework supports the development of user interfaces to operate the National Ignition Facility (NIF) using the Integrated Computer Control System (ICCS). [1] This framework simplifies UI development and ensures consistency for NIF operators. A comprehensive, layered collection of UIs in ICCS provides interaction with system-level processes, shot automation, and subsystem-specific devices. All user interfaces are written in Java, employing CORBA to interact with other ICCS components. ICCS developers use these frameworks to compose two major types of user interfaces: broadviews and control panels. Broadviews provide a visual representation of the NIF beamlines through interactive schematic drawings. Control panels provide status and control at a device level. The UI framework includes a suite of display components to standardize user interaction through data entry behaviors, common connection and threading mechanisms, and a common appearance. With these components, ICCS developers can more efficiently address usability issues in the facility when needed. The ICCS UI framework helps developers create consistent and easy-to-understand user interfaces for NIF operators.

INTRODUCTION

The ICCS User Interfaces give NIF operators access to all NIF layers, from individual devices through the highest layers of NIF workflow (referred to as the shot automation cycle). All UIs are implemented using Java version 6 and JacORB CORBA.

All user interactions with ICCS begin with a UI called the Navigator (Figure 1). On startup, the Navigator queries the ICCS configuration database for an inventory of current connection points. A tree of UIs is then presented to the user. Using search and filtering tools, operators drill down to other, more specific UIs. Adding connectivity to a new device requires only adding an entry to the ICCS configuration database.

Distributed object references for ICCS components (either actual devices or virtual software components such as the Shot Director) are obtained from a name server using the component's "taxon." A taxon contains several elements, including a component's subsystem, its location, a unique ID name, and a user-friendly alias description.

CONNECTION FRAMEWORK

UIs communicate with other ICCS components using CORBA, both for direct control and status propagation (Figure 2). [2]

Direct control is performed through the CORBA interfaces – devices can be moved, tasks can be started, data can be retrieved, etc. In most cases, UIs can control devices directly when needed.

Constantly clicking buttons to poll the NIF devices for status updates would be inefficient for NIF operators. Automatic status propagation, employing a publish/subscribe design pattern, is handled by intermediate data propagation servers, called "supervisors", which provide data sources, or "mappers", that deliver data updates to subscribers using CORBA Any type objects. When data changes occur updates propagate to all subscribed UIs which perform the appropriate data extraction and screen updates.

Supervisors reduce the stress on the Front End Processors (FEPs) by acting as a fan-out mechanism; FEPs publish status updates to only one other entity (a supervisor) rather than many (all connecting UIs).

UIs have four different connection patterns:

- Taxon Connectable: The input for connection is a single taxon. Most connectivity to individual devices is done this way, and the UIs are referred to as "control panels."

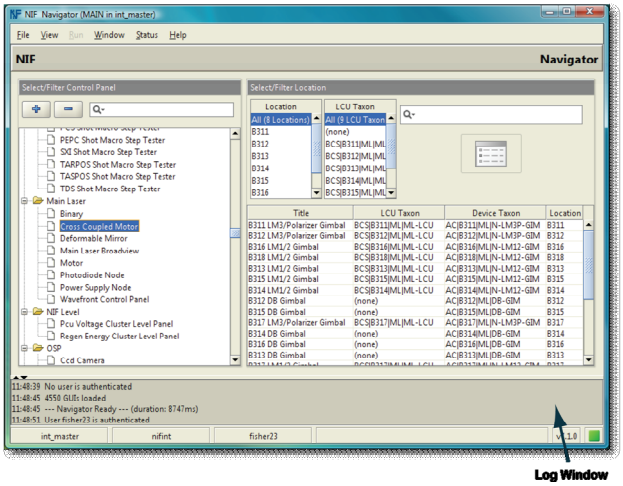

Figure 1 The Navigator serves as a starting point for NIF operators

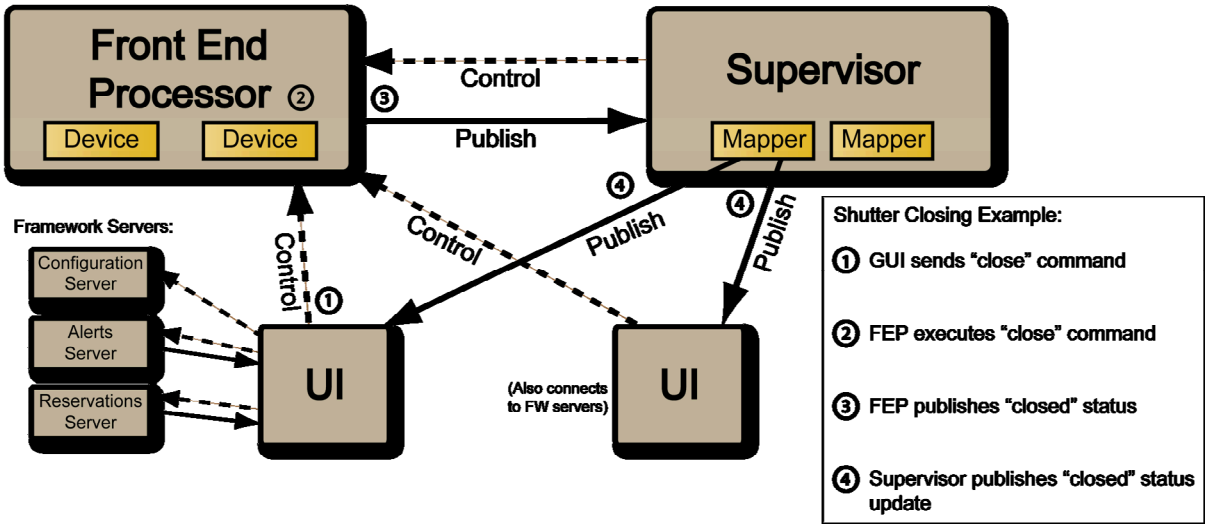

Figure 2 Communication flow between ICCS UIs and other control system components

- Location Connectable: The input for connection is a single location string, which is then used to derive a whole suite of taxons. These are generally higher level status UIs, and are referred to as "broadviews."
- Framework Service: Access to top-level framework services requires specialized UIs. Such services include system manager, reservations, alerts, and data archiving.
- Other: ICCS developed a collection of specialized UIs for performing tasks such as browsing log files and data-driven, spreadsheet-like views of arbitrary data points across the NIF facility.

DISPLAY FRAMEWORK

The ICCS UI development team created a comprehensive display framework that enables developers to create new UIs efficiently. The UI framework helps enforce a standard look and feel and includes mechanisms for safely implementing connection management, threading, and other low-level behaviors.

In addition to the standard Java Swing API, a number of third-party products are employed, including ILog JViews for interactive vector-based drawings, Jidesoft's JIDE for rich component interaction, and JFreeChart for dynamic charting.

Each UI is built on a standard base class, called a BaseDisplay. As shown in Figure 3, the BaseDisplay includes a title, status bar, and other parent behaviors such as the ability to dynamically change locations after a UI has already been displayed.

Broadviews

The broadview framework employs live, scalable schematic-like diagrams, as shown in Figure 2, that graphically depict the high level status and positions of different devices, the laser pathway, current device reservations, and any associated alerts. From these broadviews, the operator can drill down to individual device control panels. ILog JViews manages the low-level drawing behaviors. In addition to handling broadviews, JViews also handles the workflow diagrams used during the shot cycle through its hierarchical layout engine. [3]

Control Panels

Control panels provide detailed status and control of specific devices. The UI framework includes a variety of display classes, referred to as “composites,” to simplify control panel construction and insure standardization across NIF subsystems. Each composite encapsulates a

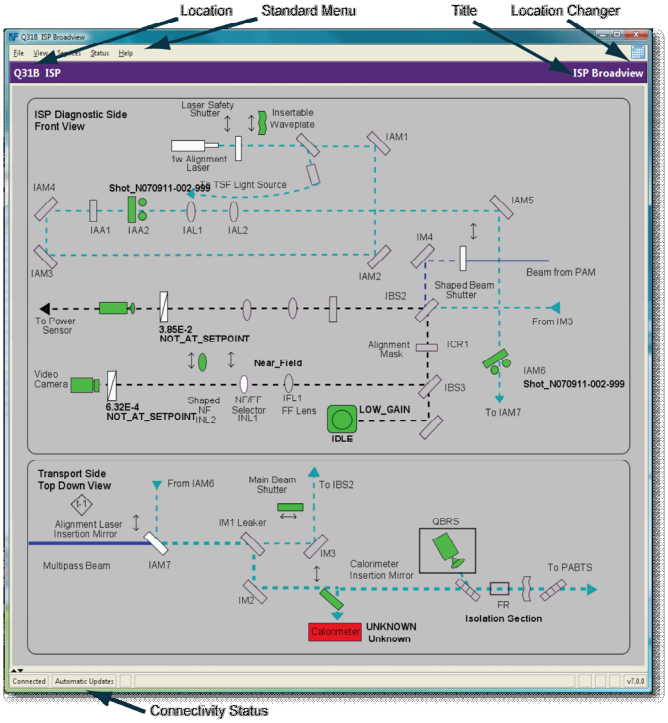

Figure 3 A broadview displays a subsystem overview, with drilldown to specific device

label, a data display mechanism, and a data input mechanism. Each composite type is specialized to handle a certain type of data display (e.g. a number, or a health status indicator) and a certain type of data input (e.g., a numeric entry field or a combo box).

One of the challenges of developing a UI solution for a control system is separating the current status of a device from the last request made against a device. For example, at a certain point in time, a motor exists at a particular position x. If the operator uses a UI to request the device move to a new position y, a long period of time could potentially pass while the device moves from x to y. Before, during, and after that process, the operators may need to know any of the following:

- The current position
- The last time the current position was updated
- The last requested position
- The time that position was requested
- The range of valid data inputs (min, max, etc)
- Any motor-level failures that may have occurred as a result of the move
- Any connectivity-level failures that may indicate the data on the UI is stale

A UI must include all the above information for every data point, but avoid clutter and usability problems. The display framework composites help to enforce the design principles and features described above. In addition, since all ICCS control panels utilize the composites extensively, usability enhancements to the composites are immediately reflected on all control panels. The composite classes come in a variety of flavors, each geared toward handling different forms of entry and display. Figure 4 depicts a sample control panel utilizing a number of composite objects. For each composite, the LED indicators report any failures, and tooltips provide details on valid data inputs and recent changes.

Threading Behaviors

In addition to handling the user I/O, the composites handle the tricky threading behaviors required for highly responsive and usable UIs. For Swing, all screen management events are handled through a single thread called the AWT (Abstract Windowing Toolkit) event thread. Any delays in processing events in this thread will degrade the usability of the UIs. Due to the highly threaded nature of CORBA, special consideration must be made to properly handle interaction with outside ICCS components.

The composite classes are designed to encapsulate threading behavior, which simplifies the implementation of high performance UIs and allows the UI implementer to focus on application-level issues.

For the automatic status updates, the model is reversed; each UI acts as a server, accepting incoming updates through a CORBA interface. To free the supervisor as soon as possible, a mapper queue is used for each UI. A separate thread processes the incoming events and performs the appropriate screen updates.

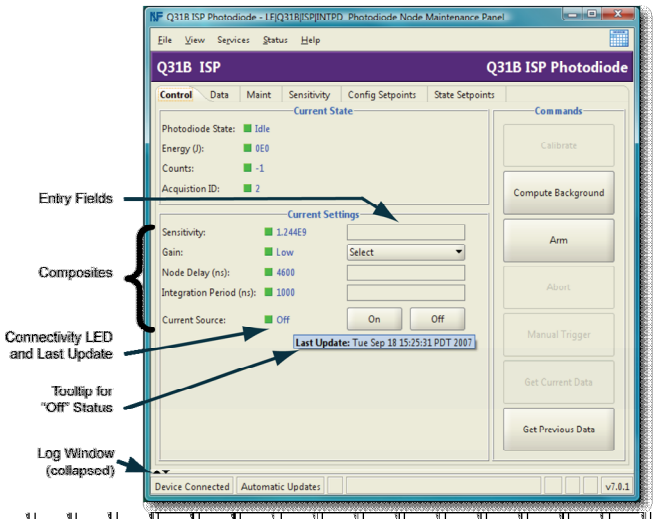

Figure 4 A control panel built on the UI Framework

The mapper content is stored in a CORBA Any, which is a general container object. It may contain one of a number of "basic" data types (for strings, floats, booleans, etc) or a custom structure (e.g., a large data array).

When a UI makes a subscription to a supervisor, it associates that subscription with a particular composite instance. If the subscription is for one of the basic map types, no further coding is required; the UI framework will automatically handle incoming updates and update the correct composite. If a custom structure is used, code for updating the composite must be written manually.

SUMMARY

The ICCS UI Framework provides building blocks used by over 120 broadviews and control panels in daily use by NIF. By focusing on a standard set of building blocks, the ICCS UIs work in a highly consistent and stable manner, and can be updated rapidly to address any usability issues that arise.

Given the generality of the ICCS UI framework, it could likely be tailored to other control system applications.

REFERENCES

[1] P. Van Arsdall, et al., "Status of the National Ignition Facility and Control System," ICALEPCS 2005

[2] R. Carey, et al., "Status of the Use of Large-scale CORBA-distributed Software Framework for NIF Controls," ICALEPCS 2005

[3] L. Lagin, et al., "Shot Automation for the National Ignition Facility", ICALEPCS 2005

*This work performed under the auspices of the U.S. Department of Energy by Lawrence Livermore National Laboratory under contract No. DE-AC52-07NA27344.